Addressing Cache Related Preemption Delay in Fixed Priority Assignment

Hai Nam Tran, Frank Singhoff, Stéphane Rubini and Jalil Boukhobza
Univ. Bretagne Occidentale, UMR 6285, Lab-STICC, F-29200 Brest, France
Email: {hai-nam.tran,singhoff,rubini,boukhobza}@univ-brest.fr

Abstract*—Handling cache related preemption delay (CRPD) in preemptive scheduling context for real-time embedded systems still stays an open issue despite of its practical importance. Indeed, classical priority assignment algorithms are only optimal when preemption costs are neglected. For example, with Audsley's Optimal Priority Assignment (OPA), as the original algorithm does not take CRPD into account, it fails frequently in identifying the schedulable task sets as it happens that the algorithm qualifies a task set to be schedulable, while it is practically not because of CRPD. In this article, we propose an approach to adapt fixed priority assignment algorithms to real-time embedded systems with cache memory. For such a purpose, we propose three extensions of the original OPA algorithm that have different degrees of pessimism, different complexities, and give different results in terms of schedulable task sets coverage. Exhaustive experimentations were achieved to evaluate the proposed approaches in terms of complexity and efficiency. The result shows that our approach provides a mean to guarantee the schedulability of the real-time embedded system while taking into account CRPD.

I. Introduction

Cache memory is a crucial hardware component used to reduce the performance gap between processor and memory. In the context of real-time embedded system (RTES), the popularization of processors with large size and multi level of caches motivates the proposition of verification methods which can handle this component [1], [2], [3], [4].

Schedulability analysis is a classical verification performed at design step of RTES. It provides means to prove that a RTES is *schedulable*. A RTES is schedulable if all its timing constraints are satisfied. To investigate schedulability, several assumptions are usually made in order to simplify the analysis. One of them is that preemption costs are negligible. *Preemption cost* is the additional time to process interrupt, manipulate task queues and actually perform context switches.

Integrating cache memory in RTES generally increases the whole performance in term of execution time, but unfortunately it can lead to an increase in preemption cost and execution time variability [5]. When a task is preempted, memory blocks belonging to the task could be removed from the cache. Once the task resumes, previously removed memory blocks have to be reloaded into the cache. Thus, a new preemption cost named *Cache Related Preemption Delay* (CRPD) is introduced. By definition, CRPD is the additional time to refill the cache with memory blocks evicted by the preemption. In [5], the author showed that the cost of context switching raises from the range of 4.2μs to $8.7\ \mu$s to the range of 38.6μs to 203.2μs when the size of the data sets of programs is larger than the cache. To sum up, integrating cache memory in RTES increases the variability of execution time and thus decreases the predictability of RTES.

978-1-4673-7929-8/15/$31.00 ©2015 European Union

Problem statement: In this article, we investigate the issues related to CRPD. First, it accounts a high proportion of the preemption cost and causes the preemption cost to become significant [5]. Second, in the context of preemptive fixed priority scheduling, there is no priority assignment algorithm which guarantees that the task set is practically schedulable when CRPD is taken into account in a given RTES. For example, with Audsley's optimal priority assignment (OPA) algorithm [6], we performed an experiment (see Fig. 3), which showed that with a high processor utilization, there is a significant gap between the number of task sets assumed to be schedulable by OPA and the number of practically schedulable task sets when considering CRPD. Indeed, without taking CRPD into account, OPA failed to identify a high number of unschedulable task sets. For instance, some of our experiments show that OPA identified 600 schedulable task sets while only 100 are practically schedulable for a 90% processor utilization (see Section VI for details about this experiment).

Contributions of the article: We propose an approach to performing priority assignment, which guarantee that a RTES is schedulable while taking into consideration CRPD. To achieve this, we extend the feasibility test proposed by Audsley [6]. The approach consists in computing the interference of both computational requirements and CRPDs when assigning a priority level to a task. The idea is to have different solutions of computing the CRPD. According to the chosen solution, the CRPD computation can be more or less pessimistic and the results in terms of schedulable task sets can be different. The work is integrated in Cheddar [7], an open source scheduling analyzer. The complexity and efficiency of the proposed solutions are evaluated with a large number of randomly generated task sets.

The rest of the article is organized as follows. Section II describes the RTES models, discusses about CRPD analysis methods, and illustrates the problem of fixed priority assignment and CRPD. Section III provides an overview of our approach. The original feasibility test in [6] is extended to take into account the CRPD. In Section IV, detailed approach and algorithms are presented. Section V discusses the complexity of the proposed solutions. In Section VI, an evaluation of our approach in terms of efficiency and complexity is given. Section VII discusses about related works and Section VIII concludes the article.

II. Background and notations

In this section, we present the background of our work. First, we present RTES model and notations used in the article. Second, an overview of the Audsley's optimal priority assignment is given. Third, we explain how CRPD is computed in existing CRPD analysis methods. Finally, we discuss about the limitation of classical fixed priority assignment algorithms.

A. RTES model and notations

We assume an uniprocessor RTES with direct-mapped cache. There are n tasks scheduled by a preemptive fixed priority scheduler. Classical notations of real-time scheduling analysis are used:

- $\tau_1, \tau_2, ..., \tau_i, ..., \tau_n$ are the tasks of the RTES.
- T_i, C_i, D_i and O_i are respectively period, capacity (or the worst case execution time), deadline, offset of task τ_i. Any task τ_i makes its initial request after O_i units of time, and then releases periodically every T_i units of time. Each release of a task is called a job. Each job requires C_i units of computation time and must complete before D_i units of time.
- $hp(i)$ (respectively $lp(i)$) is the set of tasks with higher (respectively lower) priorities than task τ_i.
- We use the term *complete priority assignment* to mention a priority ordering in which each task is assigned a priority level. To conform with the notation used in [6], priority level one is the highest priority.

B. Audsley's Optimal Priority Assignment (OPA)

The Audsley's priority assignment algorithm is optimal in the sense that for a given RTES model, it provides a feasible priority ordering resulting in a schedulable RTES whenever such an ordering exists. For n tasks, the algorithm performs at most $n \cdot (n+1)/2$ schedulability tests and guarantees to find a schedulable priority assignment if one exists.

The algorithm starts by assigning the lowest priority level n to a given task τ_i. Then, a feasibility test is used to verify if τ_i is schedulable or not. If τ_i is not schedulable at priority level n, the algorithm tries to assign the priority level n to a different task. If τ_i is schedulable, the algorithm assigns priority level n to τ_i and then, moves to the next priority level $n-1$. Then, the algorithm checks whether a task in the set of unassigned priority tasks is feasible at that priority level. The algorithm continues until all tasks are assigned a priority level. If there is not any schedulable task at a given priority level, the RTES is not schedulable and the algorithm terminates. The pseudo code for the Audsley's algorithm is given below.

```
1  for each priority level i, lowest first loop
2      for each unassigned task $\tau$ loop
3          if $\tau$ is schedulable at priority i then
4              assign $\tau$ to priority i
5              break (continue outer loop)
6          end if
7      end loop
8      return unschedulable
9  end loop
10 return schedulable
```

In order to achieve optimality, the feasibility test at line 3 must use a sufficient and necessary condition. In the original work of Audsley [6], the RTES model consists of tasks with arbitrary release times, no simultaneous release and arbitrary deadlines, the preemption cost is assumed to be zero. Indeed, the correctness and optimality in the original work of Audsley [6] depends on a specific property, which is the response time of a task is not affected by the priority ordering of higher priority tasks. This property is not true when CRPD is taken into account. Thus, we need to design an appropriate feasibility test. This feasibility test must be able to verify the feasibility of a task under a given priority level while complete priority assignment of higher priority tasks is not set. To the best of our knowledge, there is no existing work dealing with such an issue.

C. CRPD analysis methods

In this section, we explain the computation of CRPD. $\gamma_{i,j}$ is the CPRD of task τ_j preempting task τ_i. $\gamma_{i,j}$ is bounded by $BRT \cdot g$. In this computation, BRT is an upper-bound on the time required to reload a memory block in the cache (also called block reload time). g is an upper-bound on the number of cache blocks, which need to be reloaded due to preemption.

To analyze the effect of preemption on a preempted task, Lee et al. [1] introduce the concept of useful cache block (*UCB*):

Definition 1 (Useful Cache Block): A memory block m is called a useful cache block at program point P, if m may be cached at P and m may be reused at a program point P' after P that may be reached from P without evicting m.

The number of UCB at a program point P gives an upper bound on the CRPD due to a preemption at P. The maximum CRPD for a task is computed at the program point with the highest number of UCB.

Another method consists in analyzing the worst-case CRPD caused by a preempting task. Busquet et al. [2] introduce the concept of evicting cache block (*ECB*):

Definition 2 (Evicting Cache Block): A memory block of a preempting task is called an evicting cache block, if it is accessed during the execution of the preempting task.

$\mathrm{UCB_i}$ and $\mathrm{ECB_i}$ are respectively the set of Useful Cache Blocks and Evicting Cache Blocks of τ_i. In this article, we use a simplified model of UCB and ECB for a task. A task is assumed to have a fixed set of UCBs and ECBs. Any partial execution of a task uses all its ECBs and UCBs.

Considering only preempted tasks or preempting task can lead to an over-approximation. The reason is that not every UCB may be evicted during preemption, and not every ECB may evict a UCB. In addition, we have to account for the case of nested preemptions. A task τ_j can directly preempt task τ_i and also task τ_k, which was previously preempted by τ_i. In this case, we have to take into account the UCBs of both τ_i and τ_k, which are evicted by τ_j.

A method, which considers both the preempted task and the preempting task, was introduced by Tan and Mooney [3]. The CRPD is computed by taking union of all UCBs of the

$Task$	C	T	D	O	**UCB$_i$**	**ECB$_i$**	RM
A	3	20	10	0	{1}	{1,2}	3
B	5	15	8	2	{3}	{3,4,1}	2
C	2	11	11	6	∅	{2,3}	1

TABLE I: Task set example

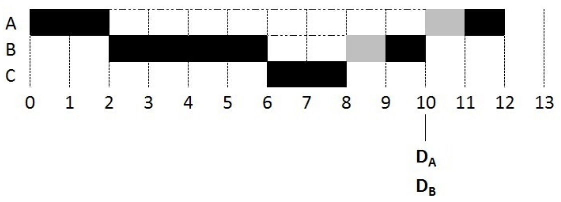

Fig. 1: Scheduling simulation with RM priority assignment policy. Task A missed its deadline. The gray block is CRPD added to the capacity of a task

preempted tasks and combined this with the set of ECBs of the preempting task.

$$\gamma_{i,j} = \text{ BRT} \cdot \left| \left(\bigcup_{\forall k \in hp(i) \cap lp(j)} \text{UCB}_k \right) \cap \text{ECB}_j \right| \quad (1)$$

Finally, Altmeyer et al. [8] introduce ECB-Unions and multi-set approaches. They gave a tighter bound on the value of $\gamma_{i,j}$. In practice, CRPD depends on memory access profile of each task and memory layout of a system. Experiments regarding CRPD computation for existing WCET benchmark could be found in [8].

D. Limitation of classical fixed priority assignment algorithm

OPA assigns a priority level to a task and verifies simultaneously the feasibility of this task. However, there is no existing feasibility test which considers CRPD that can be applied as presented in the previous sections.

Modifying priorities of tasks can minimize the CRPD and can make a task set schedulable. We give an example on how CRPD can change schedulability conditions, see Table I. We assume that $BRT = 1$ unit of time.

Let us analyze the example of Rate-Monotonic (RM) where priorities are assigned according to the period of the tasks, i.e. the lower the period of the task, the higher the priority level is. It results in a non-schedulable task set. The scheduling sequence is displayed in Fig. 1. The response time of task A is increased by 2 units of time because of the preemption cost due to task B preempting task A. In addition, the preemption cost due to task C preempting task B, which is a nested preemption, also increases the response time of task A. As we can see, the assumption of CRPD leads to a RTES where no classical priority assignment algorithms are optimal.

In this example, contrary to a Rate Monotonic priority assignment, task A, B and C must be assigned with the priority level 1,2 and 3 respectively to make the task set schedulable by totally eliminating the CRPD overhead. To conclude, one needs to take into account the CRPD early in the system model in order to verify the feasibility of tasks and adapt the priority assignment if it is necessary.

III. Problem Formulation and Overview of the approach

In this section, we present our approach and discuss problems raised. The priority assignment algorithm used is still OPA but we extend the feasibility condition to take into account the CRPD.

Regarding the feasibility condition in [6], a task is schedulable if all its jobs released during the feasibility interval can meet their deadlines. We recall that a feasibility interval is an interval for which testing of task feasibility is needed [9]. Assuming a job of task τ_i is released at time t, requires C_i units of computation time and must complete before D_i. The job experiences interference I_i^t caused by other jobs of higher priority tasks during the interval $[t, t + D_i)$. Then, the job of the task τ_i is feasible if:

$$C_i + I_i^t \leq D_i \quad (2)$$

Definition 3 (Interference): The interference that is suffered by τ_i due to higher priority tasks wishing to execute during the release of τ_i starting at t is defined as I_i^t.

I_i^t consists of three parts:

- The first part composed of both *created interference* and *remaining interference* and was solved by OPA [6]. *Created interference* is the computational requirement of higher priority tasks released in $[t, t + D_i)$. *Remaining interference* is the computational requirement of higher priority tasks τ_j that have been released before t but have not completed their execution at t. A naive approach to compute remaining interference is assessing jobs released in the interval $[0, t)$. In [6], the author provided a better approach by taking into account jobs released in the period $[t - D_i, t)$ plus the outstanding computation of the created interference of the previous period $[t - T_i, t - T_i + D_i)$. To keep it simple, we call both created interference and remaining interference as computational interfence of higher priority tasks τ_j released in the interval $[t - T_i + D_i, t + D_i)$. In the previous example with task A at priority level 3, the computational interference is the execution time of task B and C in the interval $[0, 10)$.
- Preemption cost due to task τ_j preempting task τ_i. In the previous example, it is the CRPD due to task B preempting task A.
- Preemption cost due to task τ_j preempting task τ_k that preempted τ_i, $\tau_k \in hp(i) \cap lp(j)$, i.e. preemption cost due to nested preemption. In the previous example, it is the CRPD due to task C preempting task B, where task C has higher priority level than task B.

The second and the third parts are composed of the CRPD of jobs released in the interval $[t - D_i, t + D_i)$. They are noted as CRPD Interference. The main contribution of our work consists in computing this interference. In order to have a full evaluation of the CRPD interference, one needs to evaluate: (1) the number of preemptions and (2) the preemption cost:

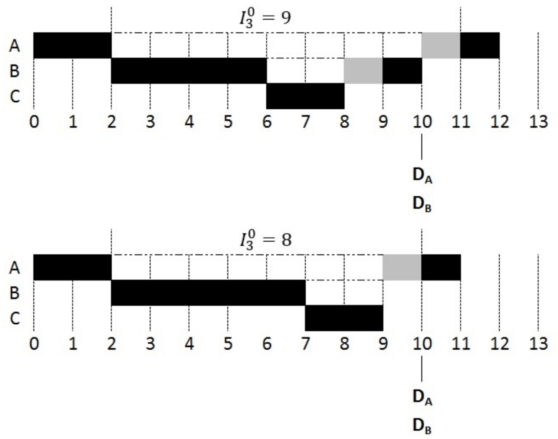

Fig. 2: Complete priority assignments of task B and C affects the computation of I_3^0

1) Number of Preemptions: With OPA algorithm, as priorities are assigned from the lowest to the highest levels, computing the number of preemptions is difficult because the priority assignments of higher priority (than τ_i) tasks is not set. In the previous example, the priorities of task B and C affect the computation I_3^0. As we can see in Fig. 2, there are two priority assignments, which results in different number of preemptions and CRPD interference. So, we need to find a solution to compute the number of preemptions with the previous constraint in mind.

2) Preemption cost - CRPD: Assume that the sets of UCB and ECB of each task are preliminary computed, then we can compute the CRPD if the preempting task and the preempted tasks are identified. However, it depends on the approaches we use to compute the number of preemptions.

In the next section, we propose three solutions un order to solve those two problems.

IV. Extending OPA with CRPD

In this section, we discuss three solutions to extend the feasibility test in [6] to take CRPD into account. The idea is to compute an upper-bound of CRPD interference. Assuming a job of task τ_i is released at time t, the CRPD interference of I_i^t is now computed by evaluating the set of jobs composed of higher priority tasks τ_j released in the interval $[t-T_i+D_i, t+D_i)$. This set is called η. In this set, the jobs are ordered by their release times in an ascending order.

A. OPA with ECB only

The first solution consists in adding the CRPD to the capacity of all higher priority tasks τ_j. The CRPD analysis using only ECB method can be used.

In this solution, we take two pessimistic assumptions: first, all activations of a task are considered as preemptions, which results in CRPD. This answers the first problem as number of preemption is equal to the number of jobs in the η. Second, the CRPD is computed by the number of ECBs of the preempting task, which is an over-approximation as presented in Section II. This answers the second problem on the cost of preemptions.

In this solution, the capacity of higher priority tasks τ_j becomes C'_j, which is computed by:

$$C'_j = C_j + BRT \cdot |ECB_j|, \forall \tau_j \in hp(i) \tag{3}$$

This solution is called OPA_CRPD-ECB. The limitation of this solution is that, an activation of a job in η set is not always a preemption. In addition, the number of evicted UCBs of preempted tasks can be lower than the number of ECBs of the preempting task. The solution is thus pessimistic for both parameters.

B. OPA with CRPD Potential preemption

The second solution consists in finding all possible preemptions and in computing the upper-bound CRPD of each preemption. This upper-bound CRPD is smaller than or equal to the number of ECB of the preempting task. This solution is less pessimistic than previous one on both parameters: the number of preemptions and preemption cost.

When complete priority assignment of higher priority tasks is not set, there is no information to decide if a task can be preempted by another task or not. We describe this problem by using the term *potential preemption*. A preemption may occur if the conditions of a potential preemption holds.

Definition 4 (Potential preemption): A potential preemption amongst jobs of tasks with no complete priority assignment is a preemption that may occur when a job is released while other jobs did not complete their execution.

We take two assumptions. First, all potential preemptions happen, in other words, every job is assumed to be able to trigger a preemption disregarding their priorities. Second, a potential preemption happens with the maximum number of preempted tasks and the maximum number of evicted UCBs.

Regarding the two problems in Section III, in this solution, the number of preemptions is equal to the number of potential preemptions. CRPD of a potential preemption is computed as follows.

In case of a potential preemption between only two jobs of τ_j and τ_k, where τ_j is released after τ_k, the CRPD is:

$$\gamma_{j,k} = \text{BRT} \cdot \mid \text{UCB}_\text{k} \cap \text{ECB}_\text{j} \mid \tag{4}$$

However, in case of nested preemptions, a job of task τ_j potentially preempts more than one job. We call $pb(j)$ the set of tasks, of which jobs are preempted by a job of τ_j. Then, the CRPD can be computed by:

$$\gamma_{pb(j),j} = \text{BRT} \cdot \left| \left(\bigcup_{\forall k \in pb(j)} \text{UCB}_\text{k} \right) \cap \text{ECB}_\text{j} \right| \tag{5}$$

The problem is to compute the set $pb(j)$. Following the second assumption, the number of elements is the maximum number of incomplete jobs at the preemption point. It can be computed by taking into account the release time and the capacity of the jobs.

The elements of the set $pb(j)$ are jobs which produce the largest set of $((\bigcup_{\forall k \in pb(j)} \text{UCB}_\text{k}) \cap \text{ECB}_\text{j})$. For example, if a job of τ_j can preempt m jobs out of n (with $m < n$), the

CRPD is computed by the combination of m jobs producing the largest set above. The computation requires a binomial coefficient complexity of $\binom{n}{n/2}$ or $\binom{n}{(n/2)+1}$.

A simplified computation could be used. In case of nested preemption, the CRPD is computed by:

$$\gamma_{pb(j),j} = \text{ BRT} \cdot \sum_{\forall k \in pb(j)} |\text{UCB}_\text{k} \cap \text{ECB}_\text{j}| \tag{6}$$

In this case, if the sets of UCBs of tasks in $pb(j)$ are mutually disjoint, Equation 6 gives the same result as Equation 5. If not, the CRPD computed by equation 6 is more pessimistic. The elements of the set $pb(j)$ are computed by evaluating m jobs with the highest number of evicted UCBs per job. The complexity in ordering the number of UCBs evicted by the preempting task of preempted tasks is $n\log(n)$.

To sum up, this solution gives a lower number of preemptions and CRPD as compared to the previous one. But still, the assumption on the number of preemption is still very pessimistic. The two versions of this solution are called OPA_CRPD-PT and OPA_CRPD-PT-Simplified.

C. *OPA with CRPD Tree*

This solution consists in computing all possible preemption sequences of jobs in η set. It consists in reducing the pessimism regarding both the number of preemptions and the cost of preemptions. The number of preemptions is reduced by considering implicit priorities between tasks to reduce potential preemptions, while the cost of preemptions is lowered by identifying the exact preempting and preempted tasks at a given preemption point. We take into account the fact that relative priorities between two tasks could be implicitly set at a potential preemption instant. If the scheduler made the decision allowing τ_j to preempt τ_k, it implicitly set the priority of τ_j higher than τ_k.

Definition 5 (Implicit priority): An implicit priority is a priority assignment of tasks undergoing a potential preemption.

This information is necessary to compute future events. For example, if the scheduler made the decision of allowing τ_j to preempt τ_k, τ_k cannot preempt τ_j in the future. In other words, the potential preemption of τ_k preempting τ_j will not happen.

To sum up, even if there is no complete priority assignment, priorities between two tasks can be set implicitly at the instant of a potential preemption. As a result, not all potential preemptions will happen.

In this solution, the job of task τ_i, which has the lowest priority, is added to the η set. We compute a tree structure to evaluate all possible preemption sequences. If there exists a branch of which the job of task τ_i is not schedulable, then the task is not schedulable at this priority level.

The tree $T = (N, E)$ is defined by N, the set of nodes and E, the set of edges. Each node n is defined by a 4-uplets (a, b, c, d) where a is a time stamp, b is the job executing at the instant a, c is the state of all jobs in the set at instant a, and d is the existing implicit priorities. The task-level priorities of jobs are set according to the scheduling decision. Each edge e from E models a scheduling decision. A scheduling decision must not violate existing implicit priorities. Branching is needed when the scheduler needs to make a decision.

Concerning the preemption cost, CRPD is computed accurately at each preemption point according to the preempting and preempted tasks.

A recursive algorithm is implemented to compute the tree. The algorithm assesses the η set. It starts from the first job and ends at the last job in the set. When the algorithm terminates, we can assess each branch by the c attribute in order to find if the job of task τ_i meets its deadline.

Regarding the two problems in Section III, in this solution, the number of preemptions of a branch is limited by the set of implicit priorities of this branch. The attribute d of each node provides information of implicit priorities. A scheduling decision must follow the set of existing implicit priorities. Each branch of the tree stores a set of consistent implicit priorities. CRPD is computed when a preemption scheduling decision happens. In this solution, the preempted tasks and preempting tasks are identified based on the attribute c of each node.

This solution is close by computing the scheduling sequence of jobs in η set with all possible priority orderings. The actual complexity highly depends on the number of potential preemptions or the number of scheduling decisions. This solution is called OPA_CRPD-Tree.

V. Complexity of the algorithms

The complexity of the algorithms lies in the computation of the interference for the jobs of higher priority tasks that are released in the interval $[t - T_i + D_i, t + D_i)$. Thus, we discuss on the complexity of the feasibility test only. In [6], the author showed that the complexity of his feasibility test is bounded by the complexity of testing task τ_n at priority level n:

$$O(X), X = \left(\frac{P_n}{T_n} \sum_{j=1}^{n-1} \left(\left\lceil \frac{T_n - D_n}{T_j} \right\rceil + \left\lceil \frac{D_n}{T_j} \right\rceil \right) \right) \tag{7}$$

In the sequel, we show how this complexity is changed due to our propositions.

A. *OPA_CRPD-ECB*

The complexity of the OPA_CRPD-ECB is the same with the complexity in [6]. OPA_CRPD-ECB only modifies the capacity of each task. No additional computation is needed.

B. *OPA_CRPD-PT*

The complexity added by this solution lies in the computation of k combinations of m potential preempted jobs. The number of combination is bounded by the binomial coefficient of n tasks.

$$O\left(\binom{n}{n/2} \cdot X \right) \tag{8}$$

The complexity of the simplified version of this solution lies in the ordering the number of UCBs evicted by the preempting task of preempted tasks. It is bounded by $n\log(n)$.

$$O(n\log(n) \cdot X) \tag{9}$$

C. *OPA_CRPD-Tree*

The tree represents all possible preemptions of a set of jobs in the interval $[t - T_i + D_i, t + D_i)$. In the worst case, computing the tree has a complexity similar to the complexity of computing the scheduling for all jobs with all possible priority assignments. Besides priority level i, there are $(i-1)$ higher priority levels. The complexity is:

$$O((n-1)! \cdot X) \tag{10}$$

In conclusion, the less pessimistic the assumptions of the solution, the higher the complexity. In the next section, we evaluate the efficiency and the complexity of those solutions.

VI. Evaluation

To evaluate the proposed approaches, experiments investigating their performances and efficiency are made. Our work is performed in the context of the Cheddar project [7]. Cheddar is an open source real-time scheduling analysis tool. Cheddar implements classical feasibility tests and scheduling algorithms for RTES. In [10], we proposed an extension of the scheduling simulator of Cheddar to RTES with caches. Cheddar has an architecture generator, which provides features to generate models of software and hardware parts of a RTES.

The configuration of our experiments is based on the existing work in [8]. Task sets are generated with the following configuration. Task periods are uniformly generated from 80 to 200 units of time. Task deadlines are implicit, i.e. $\forall i : D_i = T_i$. Processor utilization values (PU) are generated using the UUniFast algorithm [11]. Task execution times are set based on the processor utilizations and the generated periods: $\forall i : C_i = U_i \cdot T_i$, where U_i is the processor utilization of task i. Task offsets are uniformly distributed from 1 to 30 units of time.

The cache memory and cache utilization of tasks are generated with the following configuration. The cache memory is direct mapped. The number of cache blocks is equal to 16. The block-reload time is 1 unit of time. The cache usage of each task is determined by the number of ECBs. They are generated using UUniFast algorithm [11] for a total cache utilization (CU) of 5. UUniFast may produce values larger than 1 which means a task fills the whole cache. ECBs of each tasks are consecutively arranged from a cache set. For each task, the UCBs are generated according to a uniform distribution ranging from 0 to the number of ECBs times a reuse factor (RF). If set of ECBs generated exceeds the number of cache sets, the set of ECBs is limited to the number of cache sets. For the generation of the UCBs, the original set of ECBs is used.

A. *Evaluating the impact of CRPD on OPA*

The objective of this experiment is to evaluate the impact of CRPD when applying the OPA algorithm.

In each experiment, the processor utilization, which does not include preemption cost, is varied from 0.50 to 0.90 with steps of 0.05. Experiments are performed with a RF of 0.3. Task set size is fixed at 5 tasks per set. For each processor utilization value, 1000 task sets are generated.

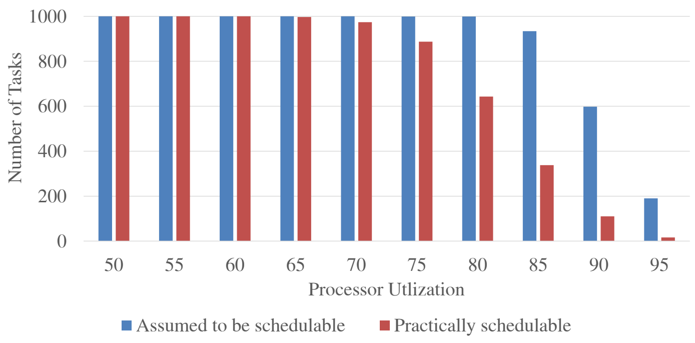

Fig. 3: Number of task sets assumed to be schedulable by OPA and number of task sets actually schedulable when CRPD is taken into account.

Fig. 3 shows the result of this experiment. For the chosen scenario, when the processor utilization is varied from 70 to 95, there is a significantly difference between the number of task sets analyzed as schedulable by OPA and the number of task sets which are actually schedulable. Without taking CRPD into account, the OPA priority assignment failed to identify significant number of unschedulable task sets. In conclusion, this experiment shows that CPRD must be taken into account when assigning priorities to tasks.

B. *Evaluating our solutions to extend OPA with CRPD*

The objective of this experiment is to evaluate the efficiency of the proposed priority assignment algorithms. Each algorithm is evaluated by two metrics. First, we evaluate the number of task sets analyzed as being schedulable by our priority assignment algorithms. Second, we evaluate how close our algorithms are to the exhaustive search approach in terms of schedulable task sets.

This experiment is composed of two steps. First, we perform priority assignments with different approaches to the generated task sets. A task set is assumed to be schedulable if the algorithm finishes assigning priorities to all tasks. Second, we perform scheduling simulations with the assigned priorities tasks to verify that the task set is practically schedulable or not while experimenting the impact of CRPD. The implementation of the scheduling simulator is detailed in [10]. Scheduling simulation is a proof that our approach is theoretically correct. In addition, we also perform an exhaustive search by testing all priority assignments for a task set and performing scheduling simulations to compare with.

Fig. 4 displays the result of this experiment. Regarding the first metric, all task sets assumed to be schedulable by the proposed approaches are by construction schedulable which is a very good result to our point of view. Indeed, the objective of this work was first to eliminate tasks sets that were found to be schedulable with OPA but that practically are not. In other words, the feasibility condition is a sufficient condition. Of course, when comparing to the optimal solution, we can see that our solutions are using some pessimistic feasibility conditions. However, the proposed priority assignment algorithms succeeded in identifying a large number of schedulable task sets. More importantly, according to the chosen solution, one can get closer to the optimal (exhaustive) solution.

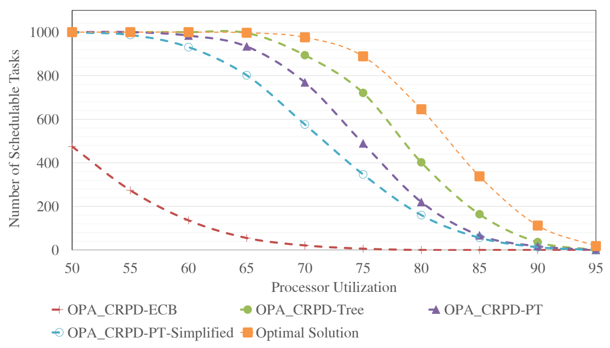

Fig. 4: Number of task sets found schedulable by the priority assignment algorithms.

Amongst the four approaches, OPA_CRPD-Tree found the highest number of schedulable task set.

The higher the processor utilization, the lower the percentage of task sets found by our approach as compared to the exhaustive search. For instance, at the processor utilization of 80%, approximately 60% of the task sets were found.

The result is compliant with the level of pessimism of each approach, as discussed in Section IV. In addition, the higher the complexity of the proposed algorithms is, the closer our approach is to the optimal solution.

Not only our approaches provide only schedulable tasks taking into account CRPD, but they also provide several task sets that were not found to be schedulable with either OPA or Rate Monotonic. However, the number of those additional task sets are not significant, only 0.7 to 1% of the generated task sets when processor utilization is greater than 70.

In conclusion, the feasibility condition used in our feasibility test is sufficient but not necessary. In fact, our priority assignment approaches succeeded in identifying large schedulable task sets. However, there is still some room for tightening the feasibility condition to get larger sets. But the major contribution is that our approaches insure to have schedulable task sets while taking the CRPD into account, to the best of our knowledge, there is no state-of-the-art study that propose such a solution.

C. Evaluating the performance of the proposed feasibility test

The objective of this experiment is to evaluate the cost of computing the interference of a set of jobs in an interval of each solution. The OPA_CRPD-ECB solution is not evaluated because it does not increase the complexity of the original solution [6] as presented in Section V.

We generated the jobs of the tasks released during an interval $[0,t)$, $t=200$. The number of tasks is varied from 2 to 30. PU is 80% and RF is 0.3. For each number of tasks, 1000 task sets are generated. Then, the computation of interference is performed. Experiments are performed on a PC with Intel Core 2 Duo CPU E8400, having 4 GB of memory, running Ubuntu 12.04 32 bits version.

The results of the experiment are shown in Fig. 5. The first observation is that computation time of OPA_CRPD-Tree increases exponentially when the number of task increase.

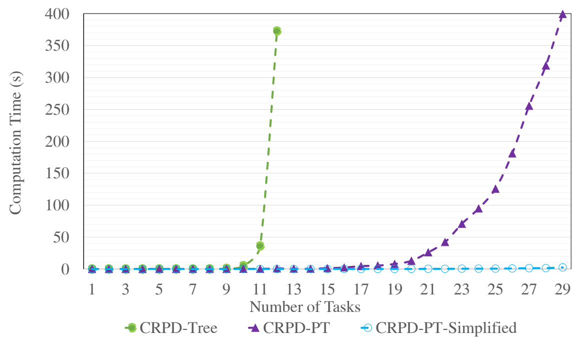

Fig. 5: Performances of the interference computation when increasing the number of tasks.

It takes averagely 370 seconds to compute the interference of 13 tasks. This is due to the exponential complexity of the feasibility test as shown in Equation 10. OPA_CRPD-PT solution has a better scalability. As an example, the computation of interference of 30 tasks takes averagely 400 seconds. This is due to the binomial coefficient complexity of the feasibility test as shown in Equation 9. The simplified version of OPA_CRPD-PT-Simplified has the best performance and scalability. The computation time for 30 tasks is less than 1 second.

In conclusion, OPA_CRPD-PT and OPA_CRPD-Tree have a significant higher complexity comparing to OPA_CRPD-PT-Simplified, which is the most pessimistic solution. Again, the higher the complexity of the proposed algorithms, the closer to the optimal solution is our approach.

VII. Related works

In this section, first, we present the related works about priority assignment algorithms. Second, we discuss about the computation of CRPD and its integration in scheduling analysis.

One of the most known priority assignment result is Rate Monotonic [12] algorithm. It showed that for synchronous periodic tasks with deadlines on requests, Rate Monotonic is the optimal priority assignment algorithm. Later, assumptions on task release times and deadlines have been relaxed. Leung and Whitehead [13] showed that Deadline Monotonic is optimal for synchronous tasks with deadlines less than or equal to their periods. Furthermore, Audsley [6] addressed asynchronous periodic tasks with arbitrary deadlines. Davis and Burns [14] improved Audsley's algorithm by taking tolerable interference into account. Their algorithm assigns priority to a task, which is not only feasible but also can tolerate highest number of interferences. Finally, Wang et al [15] proposed preemption thresholds to improve the schedulability and to reduce preemption overhead. However, in their models, the preemption cost is assumed to be zero. Dobrin et al [16] introduce a method to modify attributes of tasks and to create artifact tasks in order to reduce again the number of preemptions.

Most of the research in priority assignment algorithms on single-processor RTES have mainly focused on finding the optimal priority assignment algorithm for a specific task model. To the best of our knowledge, there is no research in the domain of priority assignment taking CRPD into account.

However, with the arrival of multiprocessor architectures in RTES, many works have been made on CRPD computation. The computation of CRPD, when a preemption occurs, is based on the two classical analysis methods called Useful Cache Blocks [1] and Evicting Cache Blocks [2]. Both of them have led to many extensions [1], [2], [3], [17]. Then, the integration of CRPD into fixed priority scheduling analysis can be made by adding CRPD into the worst case response time computation [1], [2], [18]. In their works, these authors assumed that task priorities are preliminarily assigned. Scheduling analysis with CRPD and dynamic scheduling has also been investigated by Ju et al. [19] and Luniss et al. [20]. They developed CRPD analysis for pre-emptive Earliest Deadline First (EDF) scheduling. The processor demand function in [21] was extended to take CRPD into account. In addition, feasibility tests, which took CRPD into account or not, obviously assumed that priorities of tasks are preliminarily assigned, which is not the case in this article. There are real-time scheduling simulation tools supporting models of RTES with cache memory such as Cheddar [10] and SimSo [22].

VIII. Conclusions

In this article, we presented a heuristic approach to perform priority assignment with CRPD taken into account. The problem we are dealing with is the lack of feasibility tests, which take into account the CRPD and applicable to OPA.

Our approach is based on the OPA and the original feasibility test proposed in [6]. We proposed and evaluated three solutions to extend the feasibility test in [6] to take into account the CRPD. There is a trade-off between the complexity and the pessimism of the three proposed solutions. For example, OPA_CRPD-Tree has a high complexity but find more schedulable task sets than OPA_CRPD-PT and OPA_CRPD-PT-Simplified. The feasibility conditions of our solutions are sufficient but not necessary. Indeed, task sets identified to be schedulable by our approach are always and by construction schedulable at execution time, which is not the case with the original OPA while experimenting CRPD, i.e. our approach identifies unschedulable task sets which are identified as schedulable by OPA.

This work is implemented in Cheddar real-time scheduling analyzer and is freely available at http://beru.univ-brest.fr/~singhoff/cheddar/.

There are open problems that we expect to address in the future. First, we expect to reduce the complexity of the feasibility test in the OPA_CRPD-Tree solution. Second, tightening the feasibility condition is also an important objective in order to increase the number of task sets found schedulable.

IX. Acknowledgments

Cheddar is supported by Ellidiss Technologies, Conseil régional de Bretagne, CG du Finistère, BMO, EGIDE/Campus France PESSOA number 27380SA and Thales TCS.

References

[1] C.-G. Lee, H. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *IEEE Transactions on Computers*, vol. 47, no. 6, pp. 700–713, 1998.

[2] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings, "Adding instruction cache effect to schedulability analysis of preemptive real-time systems," in *Proceedings of the 2^{nd} IEEE Real-Time Technology and Applications Symposium (RTAS)*, 1996, pp. 204–212.

[3] Y. Tan and V. Mooney, "Timing analysis for preemptive multitasking real-time systems with caches," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 1, p. 7, 2007.

[4] S. Altmeyer and C. Maiza Burguière, "Cache-related preemption delay via useful cache blocks: Survey and redefinition," *Journal of Systems Architecture*, vol. 57, no. 7, pp. 707–719, 2011.

[5] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007.

[6] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," in *Technical Report YCS 164, Dept. Computer Science*. University of York, UK, 1991.

[7] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *ACM SIGAda Ada Letters*, vol. 24, no. 4, 2004, pp. 1–8.

[8] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related preemption delay aware response time analysis for fixed priority preemptive systems," *Real-Time Systems*, vol. 48, no. 5, pp. 499–526, 2012.

[9] L. Cucu and J. Goossens, "Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors," in *Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on*. IEEE, 2006, pp. 397–404.

[10] H. N. Tran, F. Singhoff, S. Rubini, and J. Boukhobza, "Instruction cache in hard real-time systems: modeling and integration in scheduling analysis tools with AADL," in *Proceedings of the 12^{th} IEEE International Conference on Embedded and Ubiquitous Computing*, Milan, Italy, 2014.

[11] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[12] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[13] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[14] R. I. Davis and A. Burns, "Robust priority assignment for fixed priority real-time systems," in *Proceedings of the 28^{th} IEEE International Real-Time Systems Symposium (RTSS)*, 2007, pp. 3–14.

[15] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *Sixth IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 1999, pp. 328–335.

[16] R. Dobrin and G. Fohler, "Reducing the number of preemptions in fixed priority scheduling," in *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*. IEEE, 2004, pp. 144–152.

[17] S. Altmeyer, R. I. Davis, and C. Maiza, "Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," 2011.

[18] J. Staschulat and R. Ernst, "Scalable precision cache analysis for preemptive scheduling," in *ACM SIGPLAN Notices*, vol. 40, no. 7. ACM, 2005, pp. 157–165.

[19] L. Ju, S. Chakraborty, and A. Roychoudhury, "Accounting for cache-related preemption delay in dynamic priority schedulability analysis," in *Proceedings of the IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2007, pp. 1–6.

[20] W. Lunniss, S. Altmeyer, C. Maiza, and R. I. Davis, "Integrating cache related pre-emption delay analysis into edf scheduling," *University of York, York, UK, Technical Report YCS-2012-478. Available from http://www-users. cs. york. ac. uk/˜ wlunniss*, 2012.

[21] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Real-Time Systems Symposium, 1990. Proceedings., 11th*. IEEE, 1990, pp. 182–190.

[22] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche, "Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms," in *5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2014, pp. 6–p.